Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. [23 de octubre de 2008]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 25 % en operativos (Ej 3) y un 60 % sobre las preguntas de teoría (Ej 4).

[Ej. 1] **[clases (20pt)]** Escribir la implementación en `C++` del `TAD Arbol Ordenado Orientado` (clase `tree`). Para la clase `tree` implemente: `insert(p,x)`, `erase(p)` y `retrieve(p)` (u `operator*`). Para la clase `iterator` implemente `lchild()` y `right()` (u `operator++`). Observaciones:

- **Debe declarar** los **miembros privados** de las clases a declarar o implementar. Ayuda: use la figura 1.
- Si opta por la interfase *"estilo"* STL, implemente la forma **prefija** del operador `operator++` (`++p`).

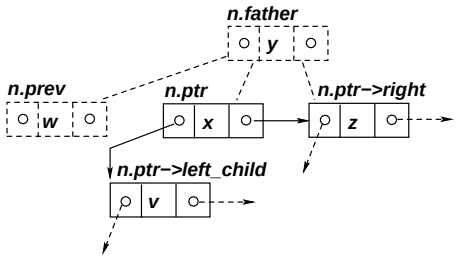

Figura 1: Entorno local de un iterator para árboles ordenados orientados.

[Ej. 2] **[Programación (total = 40pt)]**

a) **[is-full (20pt)]** Recordemos que un árbol binario (AB) es *"lleno"* si todos sus *niveles están completos (atención, no confundir con árbol completo)*. Se puede definir en forma recursiva de la siguiente manera

- El árbol vacío es lleno
- Un árbol no vacío es lleno si sus dos hijos son llenos y tienen la misma altura.

Consigna: Escribir una función predicado `bool is_full(btree<int> &T);`.
Sugerencia: Escribir una función auxiliar
`bool is_full(btree<int> &T,btree<int>::iterator n, int &height);` que retorna por `height` la altura del árbol.

b) **[max-sons-count-node (20pt)]**
Escribir una función
`tree<int>::iterator max_sons_count_node(tree<int> &T, int &nsonmax);` que, dado un árbol ordenado orientado `T` retorna la posición (o nodo o iterator) del nodo cuya cantidad de hijos es la máxima y en `nsonmax` ese número máximo de hijos. Por ejemplo, si `T=(3 (4 2 10) (6 7 (8 9 5)) (11 12 13 14 15 16))`, entonces `max_sons_count_node` debe retornar la posición (o iterator) del nodo cuyo contenido es `11` y `nsonmax=5`. Si hay más de un nodo con la cantidad máxima de hijos, entonces puede retornar cualquiera de ellos. *Sugerencia: dado un nodo debe ir recorriendo los hijos contandolos y al mismo tiempo ir reteniendo un iterator al nodo con mayor cantidad de hijos.*

[Ej. 3] [operativos (total 20pt)]

a) [**rec-arbol (10pt)**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{A, B, L, C, D, H, J, O, K, P, E, I, F, G, M\}$;
- `ORD_POST` $=\{L, B, C, O, J, P, K, H, D, I, E, F, M, G, A\}$.

b) [**huffman (10pt)**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra `WALLSTREET` $P(W) = 0.10, P(A) = 0.15, P(L) = 0.05, P(S) = 0.10, P(T) = 0.20, P(R) = 0.10, P(E) = 0.25, P(T) = 0.05$ Calcular la longitud promedio del código obtenido. Justificar si cumple o no la condición de prefijos.

[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Explique cual es la condición de códigos prefijos. De un ejemplo de códigos que cumplen con la condicion de prefijo y que no cumplen para un conjuntos de 3 caracteres.

b) Defina en forma recursiva el listado en orden previo y el listado en orden posterior de un arbol ordenado orientado con raiz `t` y sub-árboles hijos `h_1,h_2,...,h_n`.

c) Cual es el orden del tiempo de ejecución (en promedio) en función del número de elementos (n) que posee el árbol ordenado orientado implementado con celdas encadenadas por punteros de las siguientes operaciones/funciones/métodos

- `insert(p,x)`
- `begin()`
- `lchild()`
- `operador ++ (prefijo o postfijo)`
- `find(x)`
- `erase(p)`
- `*n`

d) Exprese como se calcula la longitud promedio de un código de Huffman en función de las probabilidades de cada unos de los caracteres P_i, de la longitud de cada caracter L_i para un número N_c de caracteres a codificar.

e) Para el arbol binario `(1 . (2 (3 5 .) 6))`

- como queda el árbol y que sucede si hacemos
```
btree<int>::iterator p=T.begin();
p=T.erase(p.left());
```
- y como queda el árbol asi hacemos por otro lado
```
btree<int>::iterator p=T.begin(),n;
n=p;
n=n.right();
p=T.splice(p.left(),n);
```